ПРОЕКТИРОВАНИЕ И ДЕКОМПОЗИЦИЯ ДВУНАПРАВЛЕННЫХ ПОТОКОВ ДАННЫХ ИНТЕРАКТИВНЫХ СИСТЕМ (ЧАСТЬ 1)

А.Г. ПИСКУНОВ

24 января 2009 г.

АННОТАЦИЯ

Документ содержит предложения про проектированию архитектуры интерактивных приложений и декомпозицию двунаправленных потоков данных приложение - пользователь. Разделение потоков данных по уровням абстракции позволяет выделять наиболее независимые друг от друга компоненты приложения, для проектирования которых может применяться метод STS-декомпозиции Майерса.

Содержание

1 ВВЕДЕНИЕ

В работе рассматривается вопрос анализа и декомпозиции двунаправленных потоков данных интерактивных систем. Для описания приложения, каналов данных и компонет приложения будет использоваться язык формальных спецификаций RSL (cm. [10], [9] ;).

Напомним, что в монографии [4] понятие потока данных использовалось для STS-декомпозиции на модули с заданными характеристиками, такими как прочность модуля, сцепление с другими модулями и т.д. На странице [1] мной выложен не слишком профессиональный, но, надеюсь, все таки полезный перевод 6 главы из [11] . Если под состоянием приложения понимать множество всех значений его переменных, включая входные и выходные параметры, то под потоком данных Майерс, по видимому, понимал последовательность из состояний приложения после выполнения каждого его оператора. (Обсуждение упомянутых характеристики модуля можно найти в [3] , а метод, напоминающий STS-декомпозицию Майерса, в [7]

STS-декомпозиция приложения достаточно формально приводила к разделению приложения на модули с заданными характеристиками. К сожалению, поток данных подразумевается однонаправленным - от входа приложения к выходу. Таким образом, из трех основных типов программных систем, выделенных в [8] (согласно обзору [2]):

- преобразующие системы - это системы, завершающие свое выполнение после преобразования входных данных (например, архиватор, компилятор). В таких системах обычно входные данные известны и доступны на момент запуска системы, а выходные данные доступны после ее завершения;
- интерактивные системы - это системы, взаимодействующие с окружающей средой в режиме диалога (например, текстовый редактор). Характерной особенностью таких систем является то, что они могут контролировать скорость взаимодействия с окружающей средой - заставлять окружающую среду <ждать>;
- реактивные системы - это системы, взаимодействующие с окружающей средой посредством обмена сообщениями в темпе, задаваемом средой.

STS-декомпозиция Майерса решала вопросы проектирования только первого типа систем (Условно говоря редкие входные данные - много преобразования данных).

Далее, так как реактивые системы (другой термин - системы реального времени) и интерактивные системы (системы общего назначения) с точки зрения анализа потоков данных не слишком отличаются друг от друга, Да и вообще не очень отличаются [5] , будем считать, что обсуждаются только интерактивные системы.

В случае двунаправленных потоков данных интерактивного приложения (частые входные данные - мало преобразования данных) метод STS-декомпозиции применить не удавалось.

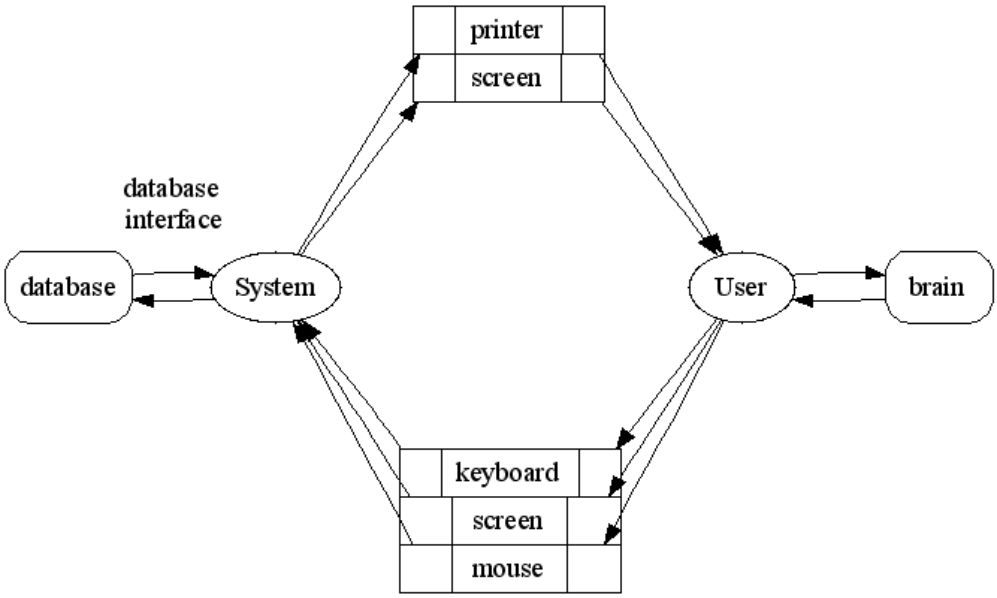

Рис. 1: Поток данных интерактивной системы 1

В случае интерактивной системы (см. 1) приложение и пользователь рассматриваются как два одновременно работающих процесса, которые обмениваются сообщениями через абстрактные каналы данных. Не слишком сложно написать самое абстрактное формальное описание работы этой пары процессов:

—- Start of Scheme

– – compile

scheme WORLD =
  class
    type
      Input, Output, DataBase – – , Memory

    channel
      keyboard, mouse, screenI : Input,
      – – special kind **of** screen like screen **of** palmPilot
      printer, screenO : Output

    variable db : DataBase

```
value
  USER :
    Unit →
      in printer, screenO out keyboard, mouse, screenI
      Unit,
  SYSTEM :
    Unit →
      in keyboard, mouse, screenI out printer, screenO
      Unit,
  userWork : Output → Input,
  systemWork : Input → write db Output

axiom
  USER() ≡ − − пусть v − результат чтения пользователем екрана
  − − компьютера
    let v = screenO? in
      ( − − screen0? − ввод с екрана в мозги пользователя
      keyboard!userWork(v)
      ⌈⌉ − − вывод рез. работы на клавиатуру
      − − ⌈⌉ операция внутреннего выбора
      − −        пользователь сам выбирает чем двигать
      − −        мышкой или клавой или экраном
      mouse!userWork(v)
      ⌈⌉ − − или кликнул мышкой
      screenI!userWork(v) − −    или тыцнул в экран
      )
    end
    [] − − операция внешнего выбора
    let v = printer? in
      ( − − копьютер заставляет пользователя
      keyboard!userWork(v)
      ⌈⌉ − − читать, то куда он вывел
      mouse!userWork(v)
      ⌈⌉
      screenI!userWork(v))
    end ;
    USER(),
  SYSTEM() ≡
    let v = keyboard? in
      ( − −    пусть v − результат опроса копьютером клавы
      screenO!systemWork(v)
      ⌈⌉ − −    результаты своей работы
      printer!
      systemWork(v) − −    вывел на экран или принтер
      )
```

```
        end
        [] -- тут пользователь заставляет
        --    компьютер читать его команды
        let v = mouse? in
          (screenO!systemWork(v) || printer!systemWork(v))
        end
        []
        let v = screenI? in
          (screenO!systemWork(v) || printer!systemWork(v))
        end ;
        SYSTEM()
    end
```

—- End Of Scheme

Более, хотя того таким образом было нетрудно описать потоки данных небольшого приложения с ограниченными набором передаваемых сообщений (см. [12]), но оказалось, что сложность описания реального интерактивного приложения возрастала слишком быстро, несмотря на использование такого высокоуровневого языка как RSL. Кроме того, разнородных данных отставалось 'слишком много' , а их преобразований 'слишком мало' для того, что появлялась возможность для STS-декомпозиции.

Главными идеями, которые позволили кратко описать интерактивное взаимодействие пары человек - компьютер, были

- выделение нескольких уровней абстракции;
- разделение двух каналов данных на несколько независимых каналов, соответствующих различным процессам;

При этом, на каждом уровне абстракции, более старший (более абстрактный) процесс управлял множеством более младших (более конкретных) процессов, но не имел доступа к их каналам данных (2).

В результате описание различных типов данных в каналах уменьшилось и появилась возможность выделения более мелких структурных элементов - компонент с однонаправленными потоками данных. Затем, к этим, формально описанные схемами языка RSL, компонетам может применяться STS-декомпозиция. Таким образом, задача проектирования интерактивного приложения оказывается сведена к уже решенной задаче.

При переходе от более абстрактного уровня спецификации (назовем схема A) к более конкретному (схема B) использовались элементы алгебраического проектирования классов ([10] , русский перевод [6]) и возможности языка RSL по автоматическому контролю соответствия следующей спецификации предыдущей (см. отношение реализации схем - implementation relation). Автоматически проверяется, что схема B содержит все сущности схемы A (типы, значения,

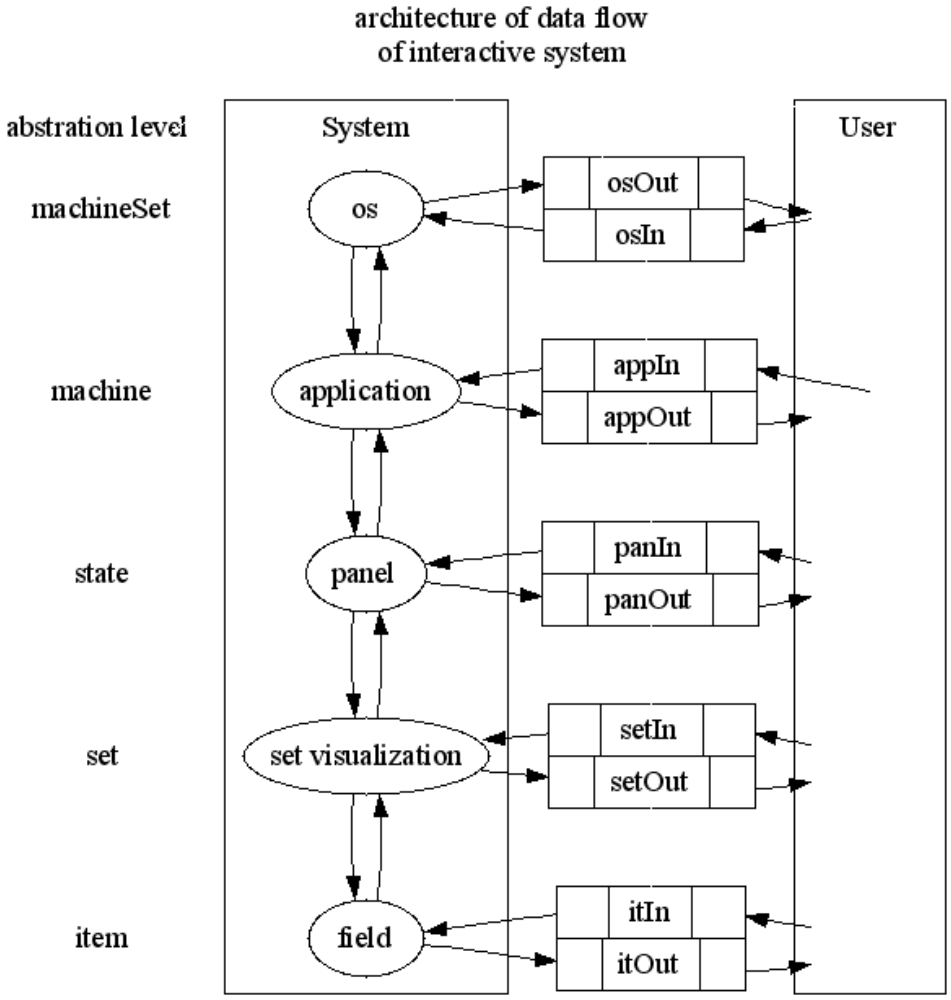

Рис. 2: Поток данных интерактивной системы 2

переменные, каналы и обьекты) с теми же именами с теми же максимальными типами. В случае обьектов - с теми же классами. (см. стр. 55, 64 (Theories and Development Relations) [10])

Отметим, что рассматривается класс интерактивных приложений со следующими ограничениями:

- оно позволяет просматривать таблицы и обзоры РСУБД;
- позволяет редактировать отдельные записи и отношения один ко многим;

- после каждой операции редактирования БД выполняет операцию commit;
- имеет мультидокументный интерфейс.

2 ПОТОКИ ДАННЫХ МЕЖДУ ПОЛЬЗОВАТЕЛЕМ И ПРИЛОЖЕНИЕМ

Для простоты будем считать, что приложение не меняет своего состояния, пока ждет команды пользователя. То есть, пользователь и группа процессов, представляющая собой приложение, работают строго последовательно: приложение получает команду от пользателя, выполняет некоторые действия, выдает результат своей работы пользователю. Результат работы приложения, кроме всего прочего, включает некоторое множество команд S, из которого пользователь может выбрать следующую команду и дать ее на выполнение приложению. Пользователь читает результат, выполняет некоторые действия над ним, выдает следующую команду (выбранную из множества S) для приложения. Процессы приложения обмениваются с пользоватем информацией через каналы данных. Считается, что пользователь может выбрать команду из множества, которое ему послало приложение. Приложение может выводить результаты своей работы в несколько каналав, пользователь читает данные изо всех каналов, а пишет команды только в один канал.

Следующее упорядоченнное множество обьектов (условное название - линейка маштабирующих обьектов) позволяет естественным образом выделить различные уровни абстракции в описании приложения:

1. операционная система (operational system) - множество автоматов (machine set);
2. приложение (application) - автомат (machine);
3. форма (панель, panel) - состояние автомата (state);
4. окно данных (record window) - визуализация множества (set);
5. поле (field)- элемент множества;

Сразу оговоримся, что обьект операционая система введен в расмотрение только для иллюстрации и законченности описания.

Введенное множество обьектов позволяет выделить уровни абстракции и, соответственно им, разделить множество программ на группы (слои) программ. Кроме этого, множество каналов данных тоже естественным образом разделятся на соответствующие группы по признаку использования одним из перечисленных обьектов:

- Уровень операционная система (множество автоматов): Операционная система имеет каналы osIn, osOut. В состояние операционной системы включается список работающих приложений.

  osIn

  через этот канал от пользователя идут команды выбора работающего приложения (следующее, предыдущее), команда на выполнение заданного приложения, команда принудительной остановки текущего работающего приложения.

  osOut

  через этот канал к пользователю идет список допустимых команд (выбрать следующее приложение, выбрать предыдущее приложение, запустить еще одно приложение, команда принудительного завершения приложения) и список работающих приложений.

- Уровень приложение (автомат): приложение имеет каналы каналы appIn, appOut. В состояние приложения включается список работающих панелей.

  appIn

  через этот канал от пользователя идут команды из главного меню приложения, команда завершения приложения, команда принудительного закрытия текущей панели.

  appOut

  через этот канал к пользователю идет список допустимых команд (выбрать следующую панель, выбрать предыдущую панель, принудительно закрыть панель, открыть еще одну панель) и список работающих панелей для выбора.

- Уровень формы (состояние автомата): каналы panIn, panOut. Формы приложения делятся на группы:

  - информационные - без объектов уровня множества;
  - табличные панели - используются для просмотра списка сущностей (немодальные).
  - панели редактирования - используются для редактирования сущности;
  - панели редактирования записи;

Данные, описывающие сущность отображаются на панели через одно или несколько окон визуализации множества (records window, окно данных) - records window.

К этому уровню относятся

- команды, изменяющие множество сущностей (задаются кпопками на панели):
- команды выбора окна визуализации множества: выбрать следующее окно данных, выбрать предыдущее окно данных.

panIn

через этот канал от пользователя идут команды из множества

PanInput = Add, Delete, Edit, Refresh, Filter, Order, Export, Exit, Cancel (Табличные панели имеет все перечисленные кнопки, а панели редактирования Add, Delete, Edit, Exit) и команды переключения окон данных.

panOut

через этот канал к пользователю идет список допустимых команд из множества PanInput. Заметим, если текущая панель была окном редактирования сущности, то список команд будет (Exit, Cancel, Edit, Delete). Если, текущая панель - окно просмотра таблицы - то список команд будет равен PanInput.

- Уровень визуализации множества: каналы setIn, setOut. изменение вида множества записей. К набору команд относятся команды изменения вида множества:
  - команды сортировки окна данных;
  - команды движения курсора по записям и колонкам окна данных;
  - команды движения окна данных по множеству записей (скроллинг);

setIn

not ready

setOut

not ready

Кроме того, приложение на этом уровне обменивается данными с базой данных (или с устройством долговременной памяти). .+ Уровень элемента множетва: через каналы передаются целые, вещественные, строковые величины.

setIn

not ready

setOut

not ready

Все приложения и панели рассматриваются как паралельно работающие процессы, которые читают и пишут из/в свои каналы. Под словом "работающий процесс"понимается, что процесс вывел в свой канал вывода свои данные (например, для операционной системы - это команды выбора приложения) и ждет одной из команд пользователя.

2.1 Основные типы для каналов данных

Тип Pointer используется для навигации в списках (в списках приложений, списках форм и т.д.), типы Direction и Zoom используются в навигации при визуализации множеств.

```
  —- Start of Scheme

scheme Types0 =
  class
    type
      Pointer ==
        first |
        prev |
        next |
        last |
        curr |
        set |
        exit |
        close,
      CloseMe
      − − to work with scrolling
      ,
      Direction == horz | vert,
      Zoom ==
        item | set | rw − − record or ∀ or set window
  end
```

—- End Of Scheme

2.2 Уровень ОС

Этот уровень абстракции как очевидный и простой рассматривается только в качестве иллюстрации.

2.2.1 Операционная система

Работа операционной системы (функция os) заключается в том, что по заданному списку приложения aps, делает вывод в свой канал. Если список aps не пуст, выполняет верхнее приложение. Читает следующую команду из своего ввода, преобразует старый список приложений aps согласно заданной команде (функция fg) и вызывает себя с новым списком приложений.

—- Start of Scheme

rsl/T0, rsl/APP_ABS

```
scheme OS_CON =
  with T0 in
  extend APP_ABS with
  class
    type
      OSInput =
        Pointer − − activate next of prev application, or close
        − − current application
        | AppPath,
      AppPath :: path : Text − − path to the new application
      ,
      OSOutput

    channel osIn : OSInput, osOut : OSOutput

    value
      os :
        Application* → in osIn, any out osOut, any Unit
      os(aps) ≡
        osOut!mkOsOutput(aps) ;
        let
          aps1 =
            if len aps > 0 then app(hd aps) ⌢ tl aps
            else aps
            end
        in
```

```
        os(fg(osIn?, aps1)) [] os(aps1)
      end
    – –   according to the specified user command.
    – –   see AppPath
    ,

    fg : OSInput × Application* → Application* ,
    mkOsOutput : Application* → OSOutput
  end
```

—- End Of Scheme

2.2.2 Абстрактное приложение

Приложение вводится как абстрактный тип. О нем известно только то, что оно есть и функция работы приложения app имеет каналы ввода вывода.

—- Start of Scheme

rsl/T0

```
scheme APP_ABS =
  with T0 in
  class
    type Application

    value
      app : Application → in any out any Application*
  end
```

—- End Of Scheme

2.2.3 Пользователь

Пользователь

- либо читает данные из канала osOut. Выполняет преобразование входных данных уровня операционной системы в выходные и выводит их в osIn;
- либо читает данные из канала osOut и выполняет функцию уровня приложения;
- ничего не делает.

Про него известно, что он может делать вывод только в один из каналов. На этой схеме этот факт не отражен.

```
  —- Start of Scheme

rsl/T0, rsl/OS_CON

scheme USR_ABS =
  with T0 in
  extend OS_CON with
  class
    value
      usr : Unit → in any out any Unit,
      usrWork : OSOutput → OSInput − − os level work
      ,
      usrWork : Unit → Unit − − app level work

    axiom
      [ usr ]
        usr() ≡
          (osIn!usrWork(osOut?) ∏ usrWork() ∏ skip) ;
          usr()
  end


—- End Of Scheme
```

2.2.4 Совместная работа пользователя и операционной системы

Совместная работа пользователя и операционной системы описывается такой схемой:

```
  —- Start of Scheme

rsl/T0, rsl/USR_ABS

scheme os_test =
  extend USR_ABS with
  class
    value
      apps : Application*

    test_case os(apps) || usr()
  end
```

—- End Of Scheme

Задаем некоторый список приложений apps и начинаем паралельно выполнять os(apps) и usr().

2.3 Уровень приложение

Ниже приводим более конкретную схему приложения, на следующем, более подробном, уровне проектирования. Более конкретное описание работы пользователя нас пока не интересует. Его можно было бы уточнять таким же способом, попутно создавая РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.

2.3.1 Конкретное приложение

Приложение является тройкой из названия приложения, списком форма и множеством команд, которые приложение может обрабатывать.

2.3.1.1 Функция работы приложения app Во первых, Фукнция закрывает приложение с пустым списком панелей. Во вторыйх, при непустом списке панелей функция заполняет канал вывода, затем вызывает функцию обработки каналов панели pan. Она получает на вход верхнюю панель и возвращает список от нуля до двух панелей. После чего канал appIn проверяется на наличие команды пользователя. Если есть ввод вызываем фукнцию focus затем опять app, иначе заканчиваем работу.

Функция focus получает команды типа Pointer или команды элемент меню MenuItem. Команды типа Pointer, которые получает приложение из канала appIn управляют навигацией форм (панелей). Аналогично схеме OS_CON.RSL, в схеме APP_CON.RSL

- по команде prev или next выбирается предыдущая или следующая панель(см. функцию focus);
- по команде close принудительно закрывается текущая панель;
- по команде exit вызывается программа save, которая сохраняет содержимое всех открытых панелей и возвращается пустой списокк;
- команды next и prev, в случае немодальной панели приводят к изменению списка напелей; Либо первый панель в списке ставится в конец, либо последняя панель ставится в начало списка.
- Если команда ввода (aCmd) может быть преобразована к типу MenuItem (mi) то по этой команде либо открывается новая панель либо соответствующая уже открытая панель выносится в голову списка pnls. Это делает функция openNewOrSelectOldTable.

```
  —- Start of Scheme

rsl/T0, rsl/PAN_ABS

scheme APP_CON =
  with T0 in
  extend PAN_ABS with
  class
    type
      AppInput =
        Pointer − −              move focus to next or prev
        − −
        |
        MenuItem − −             panel, or close current panel
      ,
      AppOutput
      − − the last parameter of application is_the main
      − − window
      ,
      Application = Text × Panel* × MenuItem-set

    channel appIn : AppInput, appOut : AppOutput

    value
      app :
        Application →
          in appIn, any out appOut, any Application*
      app(nm, pnls, ms) ≡
        if len pnls > 0
        then
          appOut!mkAppOutput(nm, pnls, ms) ;
          let pnl = pan(hd pnls) in
            app(nm, focus(appIn?, pnl ⌢ tl pnls), ms)
            □
            ⟨ (nm, pnl ⌢ tl pnls, ms)⟩
          end
        else ⟨ ⟩ − − closed
        end,

      focus :
        AppInput × Panel* → in any out any Panel*
      focus(aCmd, pnls) ≡
        let l = len pnls in
          if l ≤ 0 then ⟨ ⟩
          elsif aCmd = close
            then  − −
```

```
            tl pnls
        elsif aCmd = exit then save(pnls) ; ⟨ ⟩
        elsif isModal(hd pnls) then pnls
        elsif aCmd = prev ∧ l > 1
          then
            ⟨ pnls(l)⟩ ⌢ ⟨ pnls(x) | x in ⟨ 1 .. l − 1⟩ ⟩
        elsif aCmd = next ∧ l > 1
          then ⟨ pnls(x) | x in ⟨ 2 .. l⟩ ⟩ ⌢ ⟨ pnls(1)⟩
        else
          case aCmd of
            AppInput_from_MenuItem(mi) →
              openNewOrSelectOldTable(mi, pnls),
            _ → pnls
          end
        end,

    mkAppOutput : Application → AppOutput
end
```

—- End Of Scheme

2.3.2 Проверка соответствия абстрактного и конкретного описания приложения

Для доказательства того, что все свойства абстрактной схемы app_abs.rsl наследуются (реализуются, implement) подробной схемой app_con.rsl используется следующая схема

—- Start of Scheme

rsl/APP_ABS, rsl/APP_CON

devt_relation
app_test(APP_CON **for** APP_ABS) : |– APP_CON $\preceq$ APP_ABS

—- End Of Scheme

2.3.3 Абстрактная форма

Задано, что панель может быть модальной (модальная панель не позволяет выбор других панелей приложения) или не модальной (немодальная панель позволяющяет выбор других панелей приложения);

```
  —- Start of Scheme

rsl/T0

scheme PAN_ABS =
  with T0 in
  class
    type Panel, MenuItem

    value
      pan : Panel → in any out any Panel* ,
      openNewOrSelectOldTable :
        MenuItem × Panel* → in any out any Panel* ,
      save :
        Panel* →
          out any Unit − − to save state of the panels
      ,
      isModal : Panel → Bool
  end


—- End Of Scheme
```

3 ЗАКЛЮЧЕНИЕ

С окончательным описанием работы интерактивной системы на уровне приложения закончим первую часть.

Список литературы

[1] Г. Майерс. Надежность Программного Обеспечения: Композиционное проектирование приложения. http://users.iptelecom.net.ua/~agp1/ru/g.myers.html.

[2] Шопырин Д.Г. and Шалыто А.А. "Синхронное программирование" , 2004. http://www.softcraft.ru/auto/switch/syncprog/index.shtml.

[3] Технология программирования. Лекция 7. http://www.ergeal.ru/txt/archive/cs/tp/07.htm.

[4] Г. Майерс. Надежность программного обеспечения, 1980.

[5] Горошко Егор? Real-time os - религиозная война компьютерной индустрии или заметки на полях одной статьи, 2003. http://qnxclub.net/files/articles/RemarksOnTheMargins/RemarksOnTheMargins.html.

[6] А.Г. Пискунов. The RAISE Method Group: АЛГЕБРАИЧЕСКОЕ ПРОЕКТИРОВАНИЕ КЛАССА, 2007. http://users.iptelecom.net.ua/~agp1/ru/ClassDesign.html.

[7] С.Орлов. Технологии разработки программного обеспечения: Учебник для вузов, 2003. http://www.tsi.lv/books/orlov_annotation.pdf.

[8] Harel D. and Pnueli A. On the development of reactive systems, 1985.

[9] Chris George. Introduction to RAISE, 2002. http://users.iptelecom.net.ua/~agp1/arts/RAISE4.pdf.

[10] Chris George. Introduction to RAISE. UNU-IIST report No. 249, 2002. http://users.iptelecom.net.ua/~agp1/arts/report249.pdf.

[11] G. J. Myers. Software reability. principles and practices, 1976.

[12] A.G. Piskunov and V.L. Ilyuhin. The usage of formal specification languages for the design of rdbms. http://users.iptelecom.net.ua/~agp1/ru/gsau.html.